COL 724 Assignment 3

Nikhil Unavekar
2020CS10363
Nov 5 2023

➔Topology (topology.py) -

It defines a custom network topology using the Mininet network emulator. Mininet allows you to create virtual network topologies for testing and experimenting with SDN (Software-Defined Networking) concepts. In this case, the custom network topology consists of hosts, switches, and links with specified characteristics.

Here's an explanation of the topology:

Import necessary modules from Mininet:

Topo: This is used to define the network topology.
TCLink: This is used to create links between network elements with specific characteristics.
RemoteController: This is used to define a remote SDN controller.
Define a class named CustomTopology that extends Topo. This class represents the custom network topology.

Inside the build method of CustomTopology, the following elements are defined:

Two switches, s1 and s2, are created using the addSwitch method.

Three hosts, h1, h2, and h3, are connected to s1. Each host is assigned a MAC address using the addHost method, and links are added with specified bandwidth (bw) and delay using the addLink method. These hosts are connected to s1 and have MAC addresses "00:00:00:00:00:01", "00:00:00:00:00:02", and "00:00:00:00:00:03".

Two hosts, h4 and h5, are connected to s2. Similar to the hosts connected to s1, these hosts are assigned MAC addresses and linked to s2 with specified bandwidth and delay. Their MAC addresses are "00:00:00:00:00:04" and "00:00:00:00:00:05".

A link is created between switches s1 and s2 to connect them.

The controllers dictionary is defined to specify the controller for the topology. In this case, a RemoteController is used, which implies that there should be a remote SDN controller running, and this topology will connect to it.

The topos dictionary is defined to specify the topology that Mininet should use. The "customTopology" key is associated with a lambda function that creates an instance of the CustomTopology class when the topology is instantiated.

➔Controller Hub (controller_hub.py)-

- The Python code is a controller application for software-defined networking (SDN) using the Ryu framework. Ryu is a popular SDN controller framework written in Python.

- Here's a brief description of the code:

    - Import necessary modules and libraries, including the sys module, collections, and Ryu-related modules.

    - Check the Python version to determine whether to import MutableMapping from collections.abc or just collections. This is done to handle differences in Python versions.

    - Define a class named Controller that extends RyuApp. This class represents the SDN controller application.

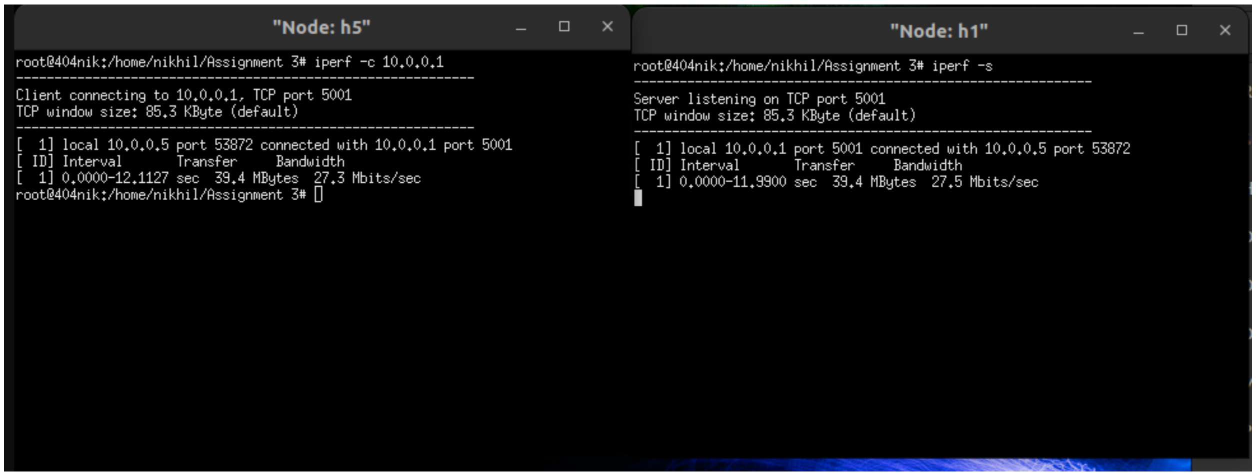
    - Specify the OpenFlow protocol version(s) supported by the controller. In this case, it's set to OpenFlow 1.3.

    - Define an __init__ method for the Controller class, which initializes the controller application.

    - Define two event handlers using the @set_ev_cls decorator:

    - features_handler: This handler is called when the controller receives a switch features request response. It installs a flow table modification to push packets to the controller, acting as a rule for flow-table misses.

- packet_in_handler: This handler is called when a packet is sent to the controller. It processes the incoming packet and floods it out to all ports.
- The __add_flow method is used to add flow table entries to the switch, which defines how packets should be forwarded based on certain criteria.



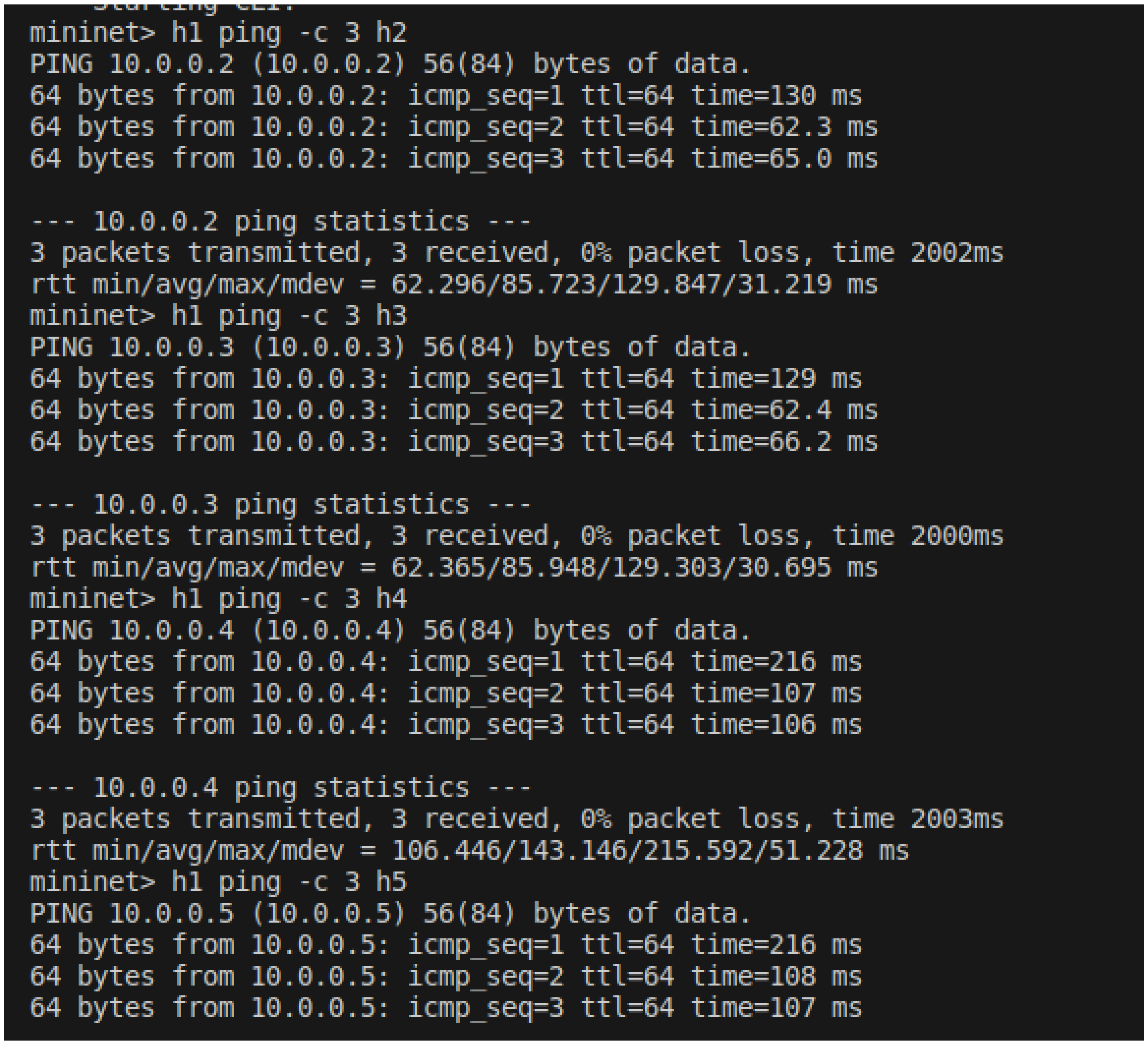
"Node: h5"

```
root@404nik:/home/nikhil/Assignment 3# iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.5 port 53872 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-12.1127 sec  39.4 MBytes  27.3 Mbits/sec
root@404nik:/home/nikhil/Assignment 3# 
```

"Node: h1"

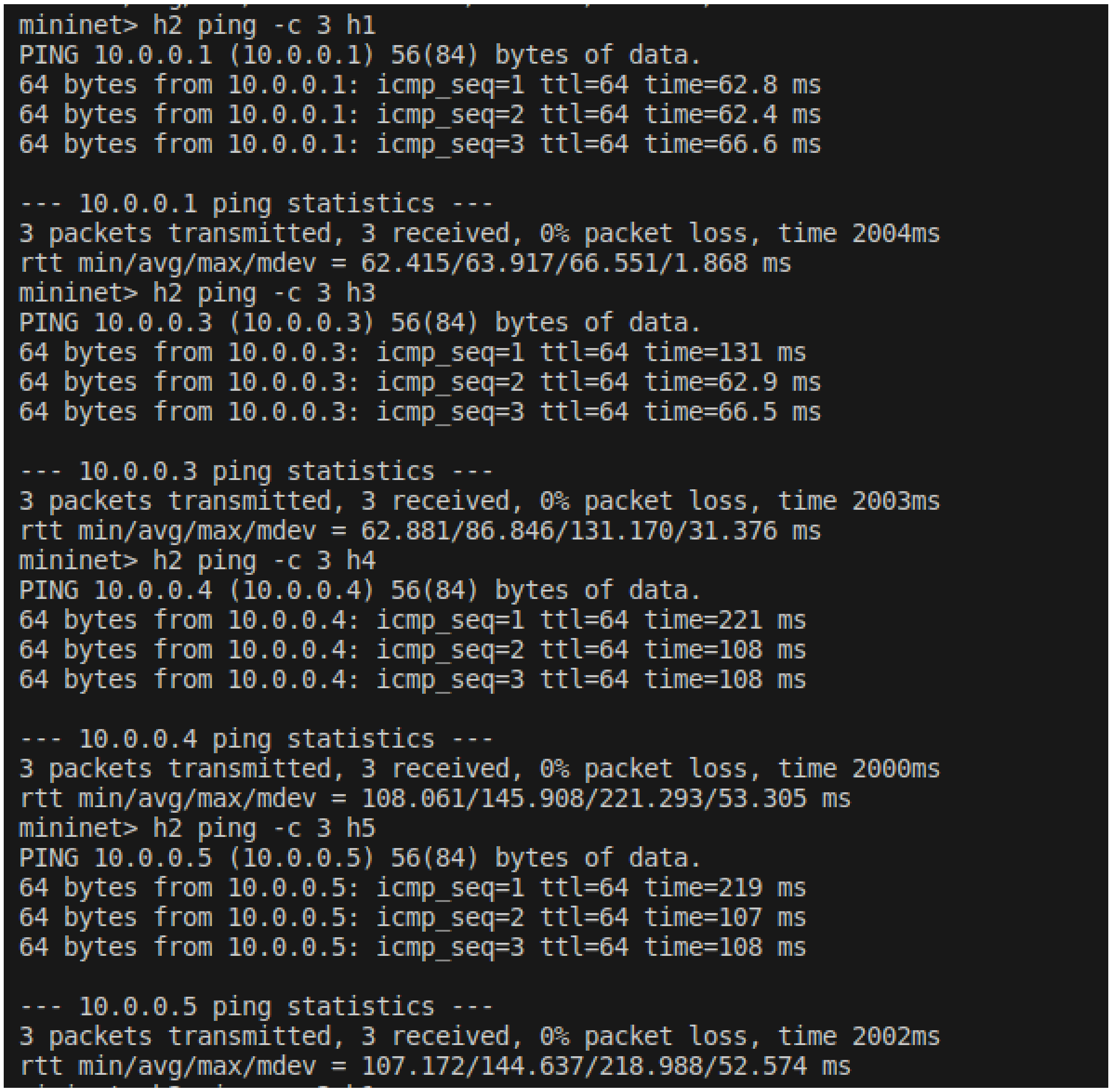
```
root@404nik:/home/nikhil/Assignment 3# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.1 port 5001 connected with 10.0.0.5 port 53872
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-11.9900 sec  39.4 MBytes  27.5 Mbits/sec
```

Bandwidth h1_h5 in controller hub



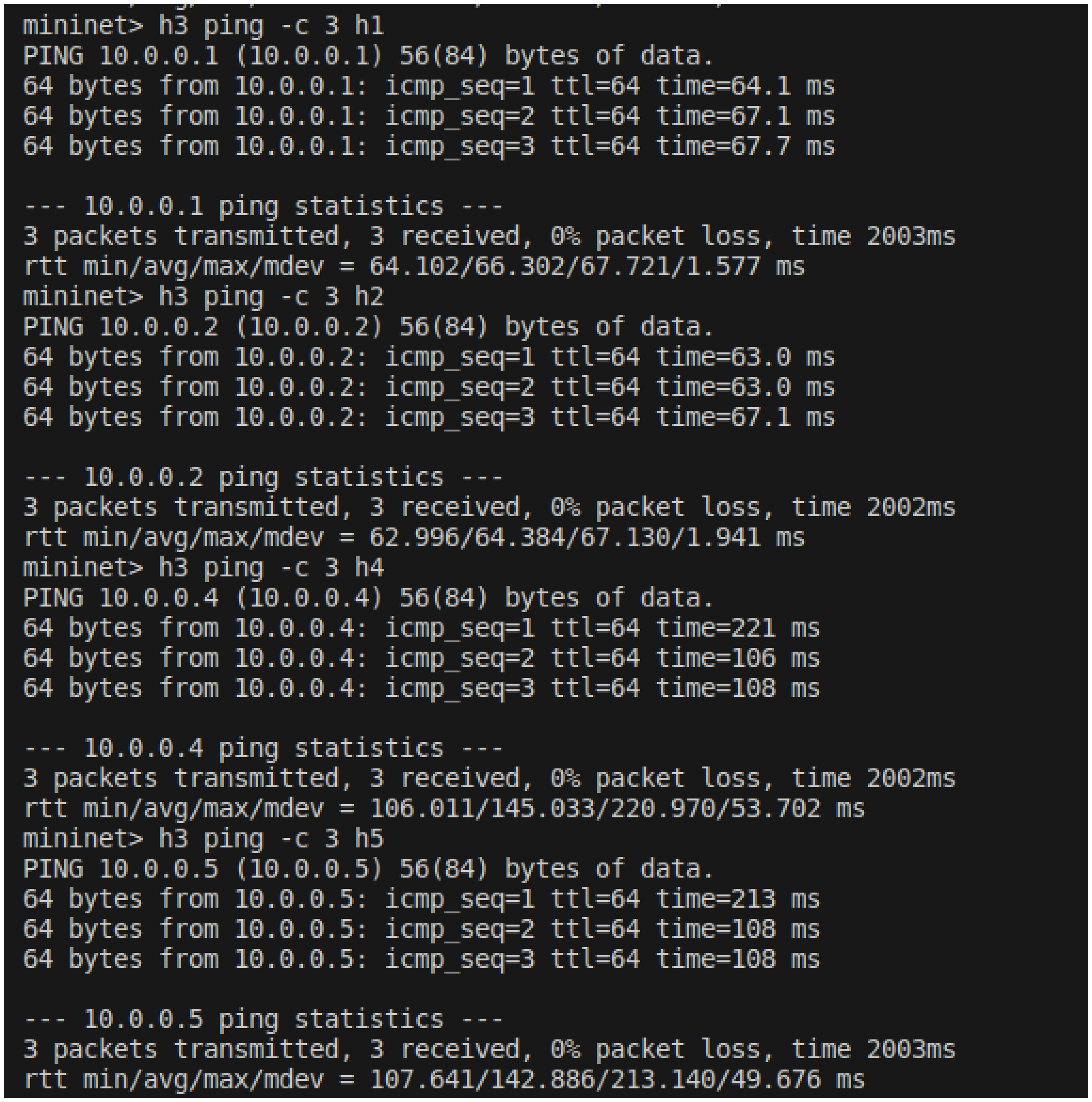
```
mininet> h1 ping -c 3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=130 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=62.3 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=65.0 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 62.296/85.723/129.847/31.219 ms
mininet> h1 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=129 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=62.4 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=66.2 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 62.365/85.948/129.303/30.695 ms
mininet> h1 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=216 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=107 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=106 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 106.446/143.146/215.592/51.228 ms
mininet> h1 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=216 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=108 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=107 ms
```

Ping tests



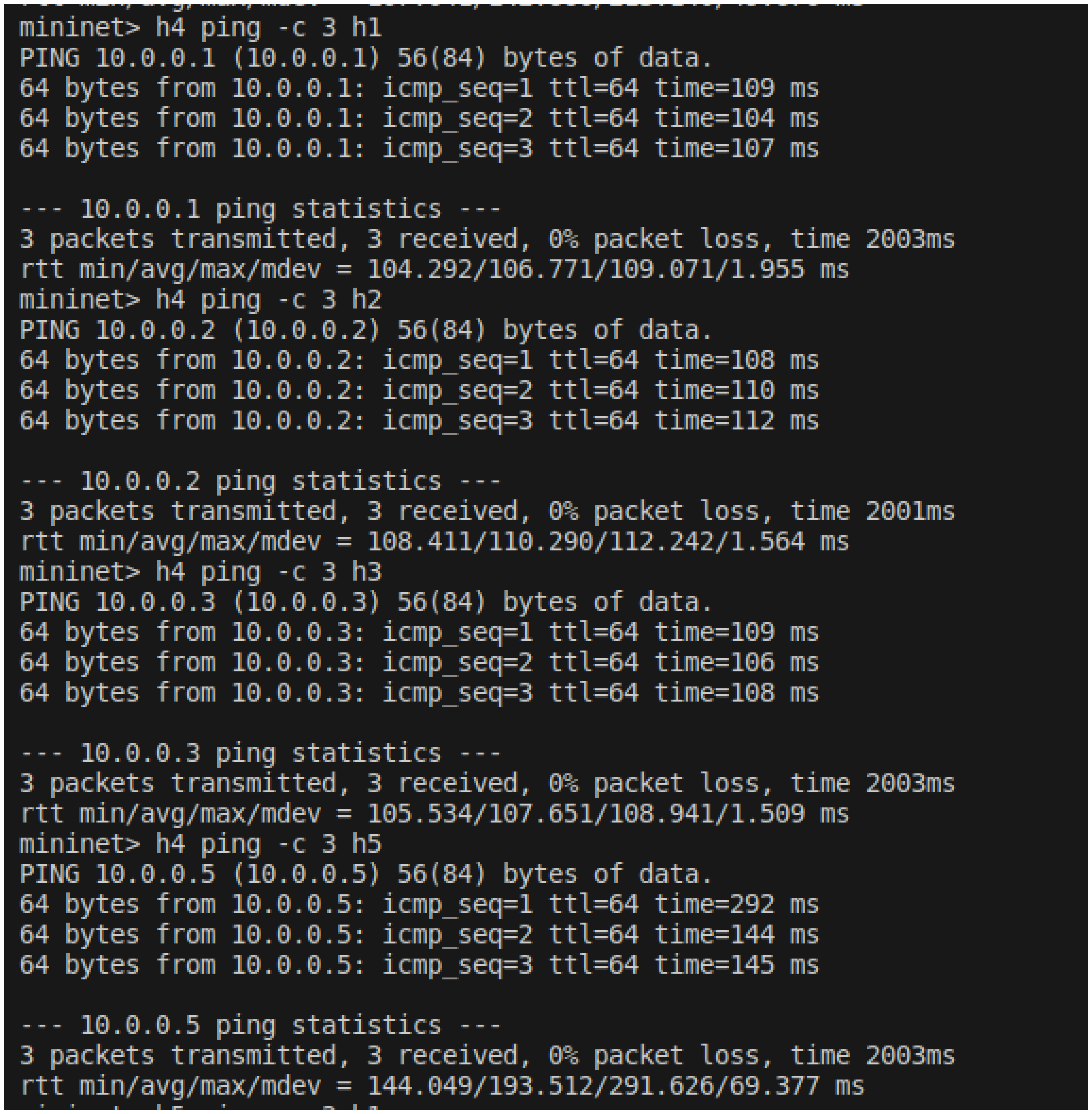
```
mininet> h2 ping -c 3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=62.8 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=62.4 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=66.6 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 62.415/63.917/66.551/1.868 ms
mininet> h2 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=131 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=62.9 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=66.5 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 62.881/86.846/131.170/31.376 ms
mininet> h2 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=221 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=108 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=108 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 108.061/145.908/221.293/53.305 ms
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=219 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=107 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=108 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 107.172/144.637/218.988/52.574 ms
```

Ping tests



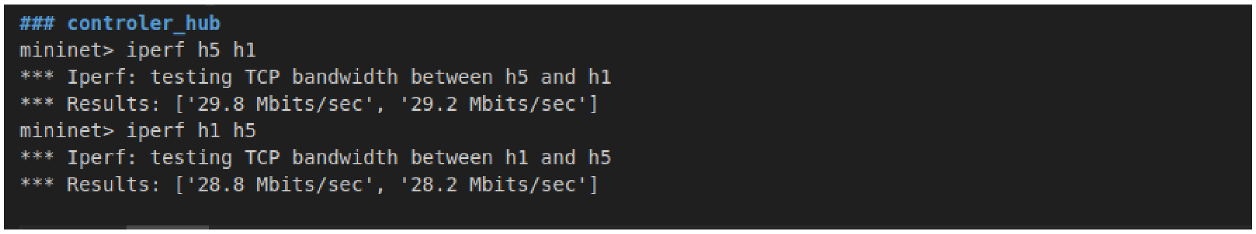
```
mininet> h3 ping -c 3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=64.1 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=67.1 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=67.7 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 64.102/66.302/67.721/1.577 ms
mininet> h3 ping -c 3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=63.0 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=63.0 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=67.1 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 62.996/64.384/67.130/1.941 ms
mininet> h3 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=221 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=106 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=108 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 106.011/145.033/220.970/53.702 ms
mininet> h3 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=213 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=108 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=108 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 107.641/142.886/213.140/49.676 ms
```

Ping tests



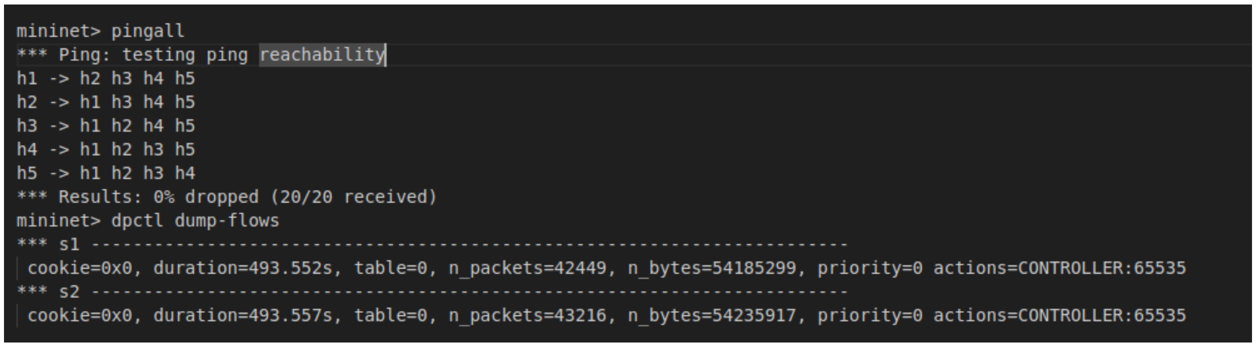
```
mininet> h4 ping -c 3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=109 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=104 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=107 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 104.292/106.771/109.071/1.955 ms
mininet> h4 ping -c 3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=108 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=110 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=112 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 108.411/110.290/112.242/1.564 ms
mininet> h4 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=109 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=106 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=108 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 105.534/107.651/108.941/1.509 ms
mininet> h4 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=292 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=144 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=145 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 144.049/193.512/291.626/69.377 ms
```

Ping tests



```
### controler_hub
mininet> iperf h5 h1
*** Iperf: testing TCP bandwidth between h5 and h1
*** Results: ['29.8 Mbits/sec', '29.2 Mbits/sec']
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['28.8 Mbits/sec', '28.2 Mbits/sec']
```

Iperf



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=493.552s, table=0, n_packets=42449, n_bytes=54185299, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=493.557s, table=0, n_packets=43216, n_bytes=54235917, priority=0 actions=CONTROLLER:65535
```

Ping all and installed rules

➔Learning Switch(learning_switch.py) -

- This is modified code is still an SDN controller application using the Ryu framework, but it incorporates packet learning functionality. It allows the controller to learn and remember the association between MAC addresses and the corresponding switch ports, which can be used for forwarding packets efficiently.

- Here's an overview of the modifications and the learning mechanism:

  - Import additional packet protocols such as ethernet, ipv6, and lldp to handle different packet types.

  - Define a class named Controller that extends RyuApp. This class represents the SDN controller application.
  -
  - Specify the OpenFlow protocol version(s) supported by the controller, which is set to OpenFlow 1.3.
  -
  - Define a list called ILLEGAL_PROTOCOLS, which includes packet protocols that you consider illegal or want to filter out. These protocols are checked in the __illegal_packet method.
  -
  - In the __init__ method, initialize the controller application. It creates an empty dictionary called mac_port_map, which will store MAC address to port mappings.

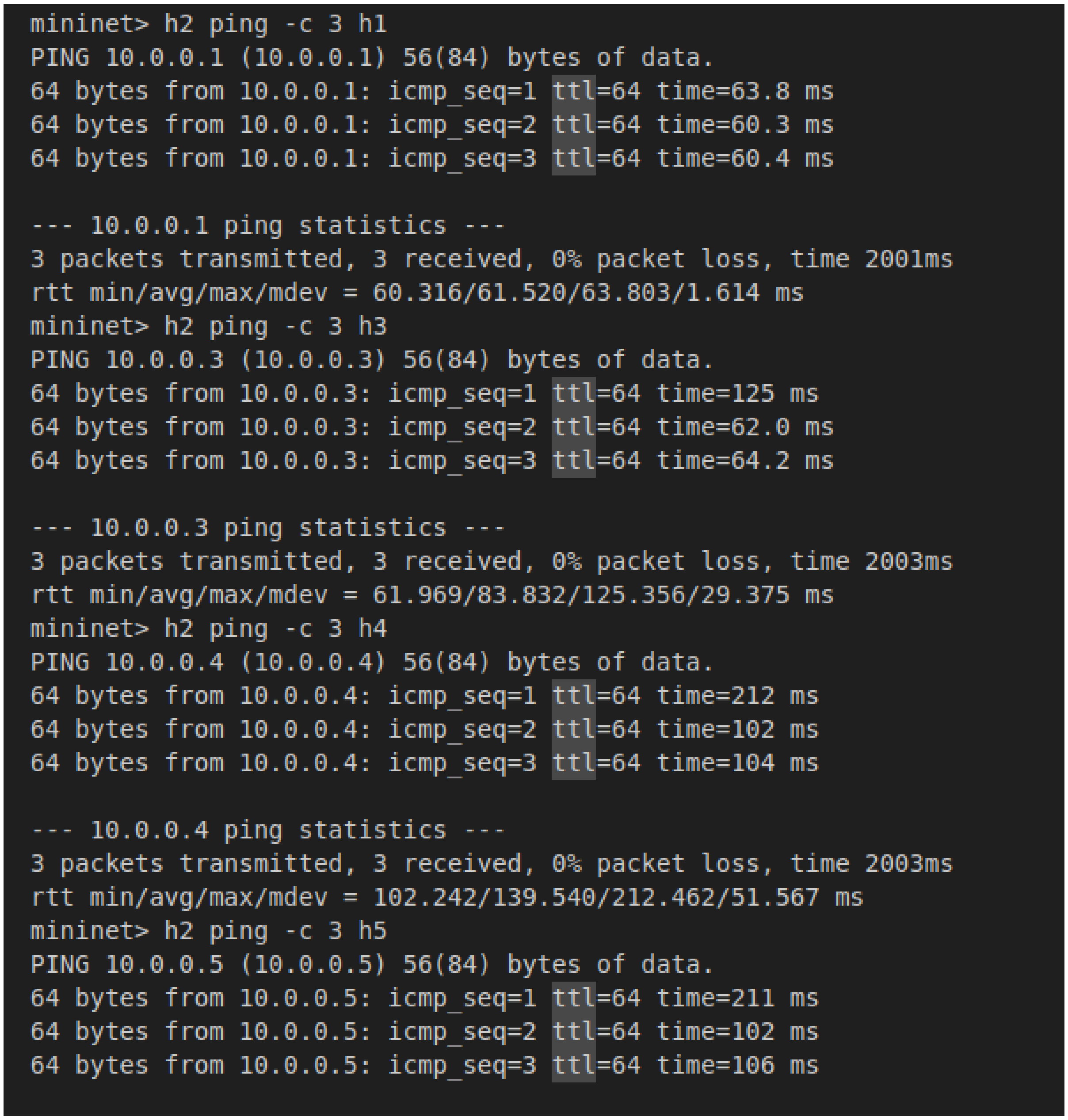
- The features_handler method is called when a switch connects to the controller (during the handshake). It sets up an initial flow entry to send packets to the controller and initializes the mac_port_map for the connected switch.

- The error_msg_handler method handles OpenFlow error messages received from the switch and logs information about the error.

- The packet_in_handler method processes incoming packets from the switch:

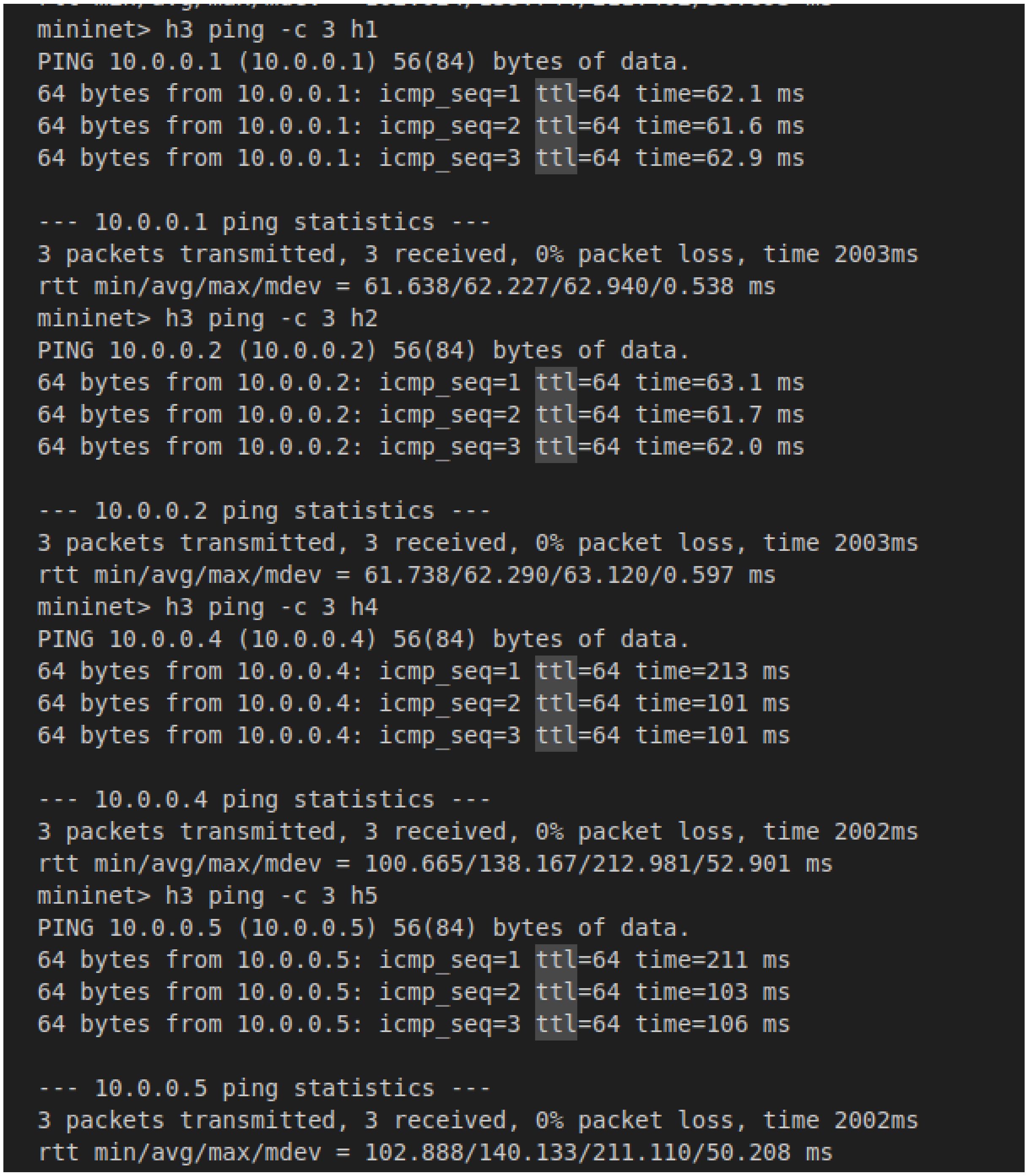
- It checks if the incoming packet contains any illegal protocols (e.g., IPv6 or LLDP) and filters them out.
- If the packet has an Ethernet header, it learns the source MAC address and the input port.
- If the destination MAC address is known in the mac_port_map, it sets the output port accordingly. It also installs a flow entry in the switch's flow table for efficient future forwarding.
- If the destination MAC address is unknown, it floods the packet out to all ports (except the input port).
- The __illegal_packet method checks if the packet contains any illegal protocols from the ILLEGAL_PROTOCOLS list and returns True if it finds any, allowing you to log and filter out such packets.

- The __add_flow method is used to add flow table entries to the switch. It allows the controller to define how packets should be forwarded based on certain criteria. In this case, it's used to add entries for MAC address-based forwarding.



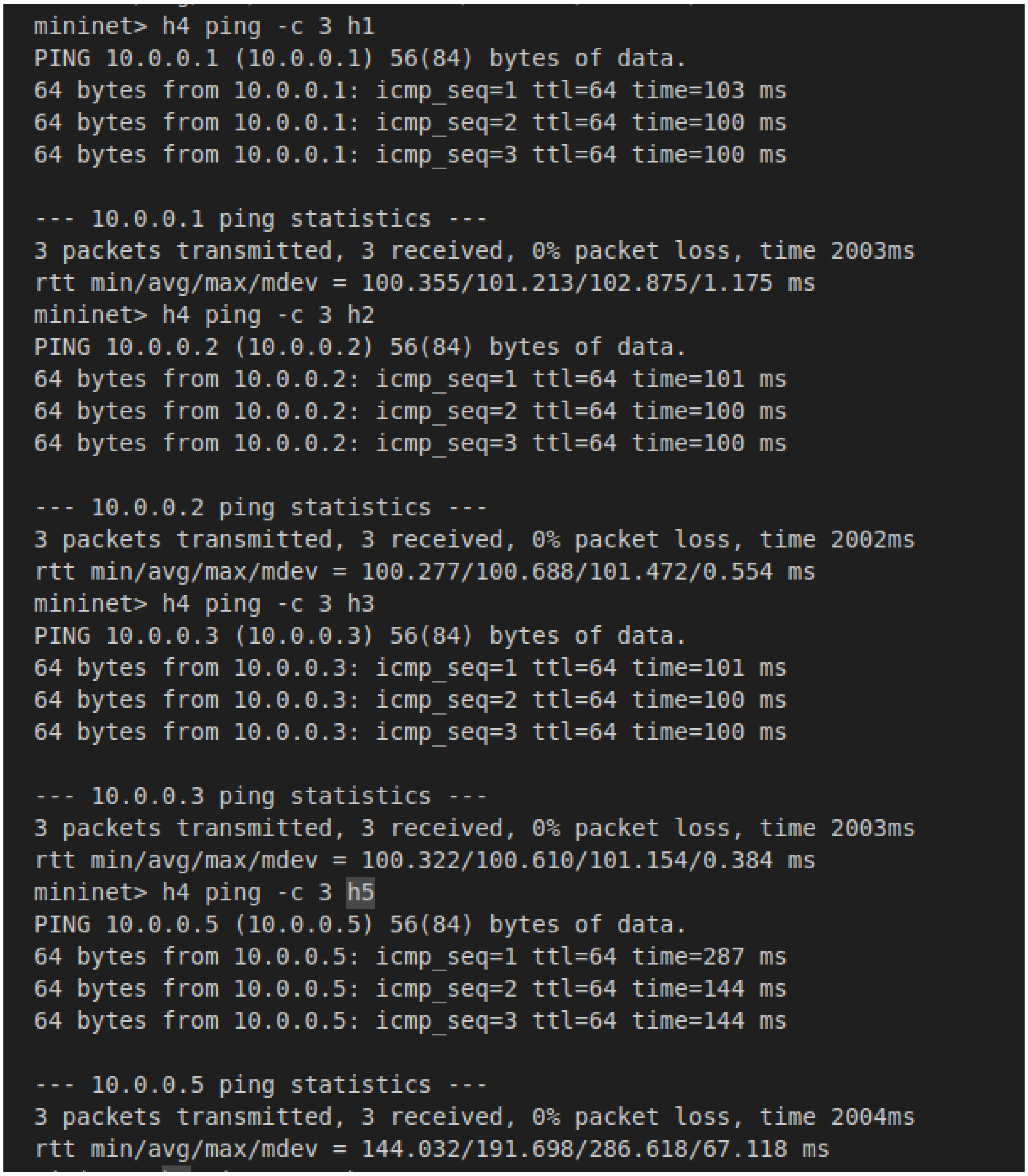
```
mininet> h2 ping -c 3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=63.8 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=60.3 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=60.4 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 60.316/61.520/63.803/1.614 ms
mininet> h2 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=125 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=62.0 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=64.2 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 61.969/83.832/125.356/29.375 ms
mininet> h2 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=212 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=102 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=104 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 102.242/139.540/212.462/51.567 ms
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=211 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=102 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=106 ms
```

Ping each other



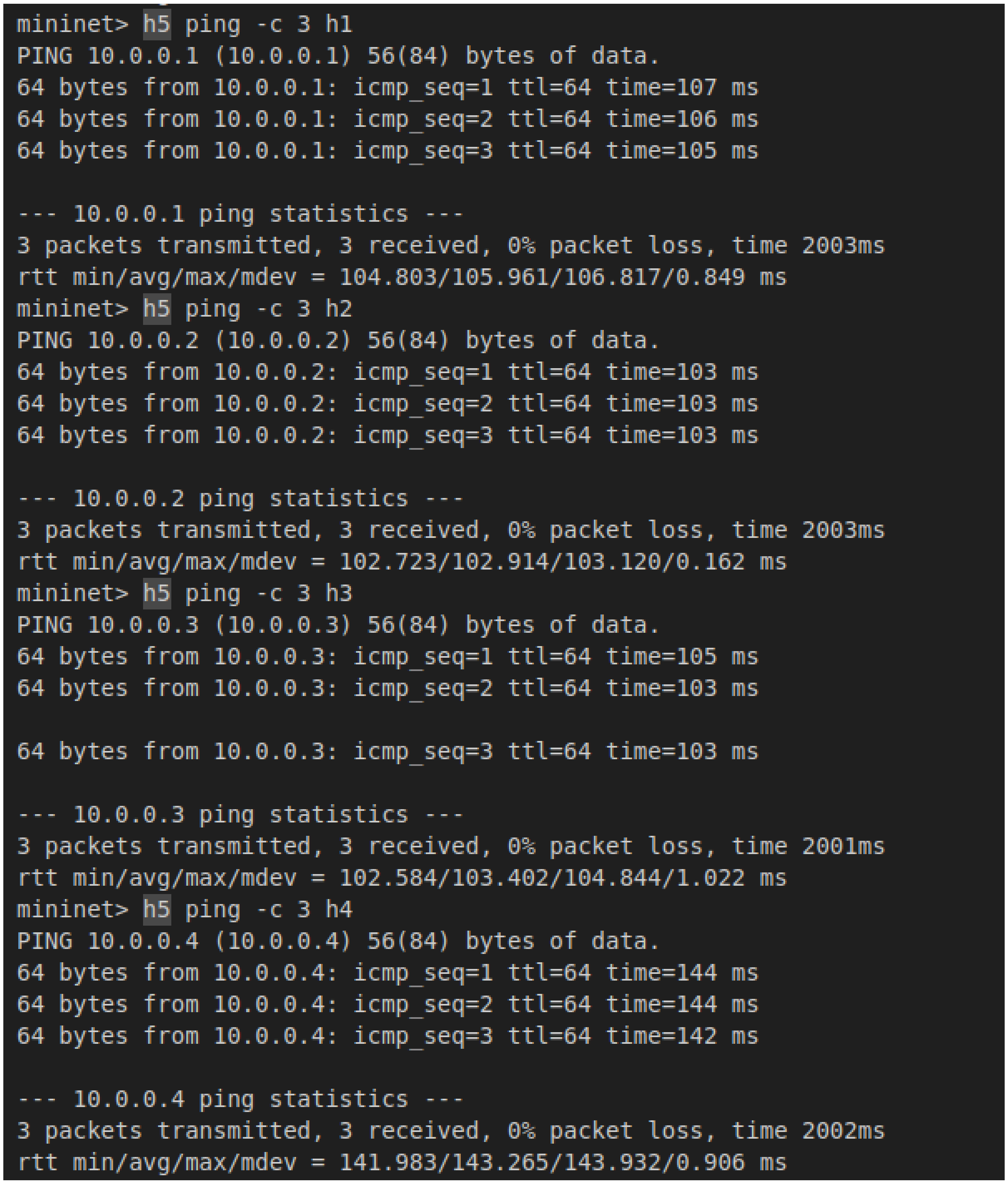
```
mininet> h3 ping -c 3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=62.1 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=61.6 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=62.9 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 61.638/62.227/62.940/0.538 ms
mininet> h3 ping -c 3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=63.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=61.7 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=62.0 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 61.738/62.290/63.120/0.597 ms
mininet> h3 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=213 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=101 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=101 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 100.665/138.167/212.981/52.901 ms
mininet> h3 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=211 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=103 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=106 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 102.888/140.133/211.110/50.208 ms
```

Ping each other



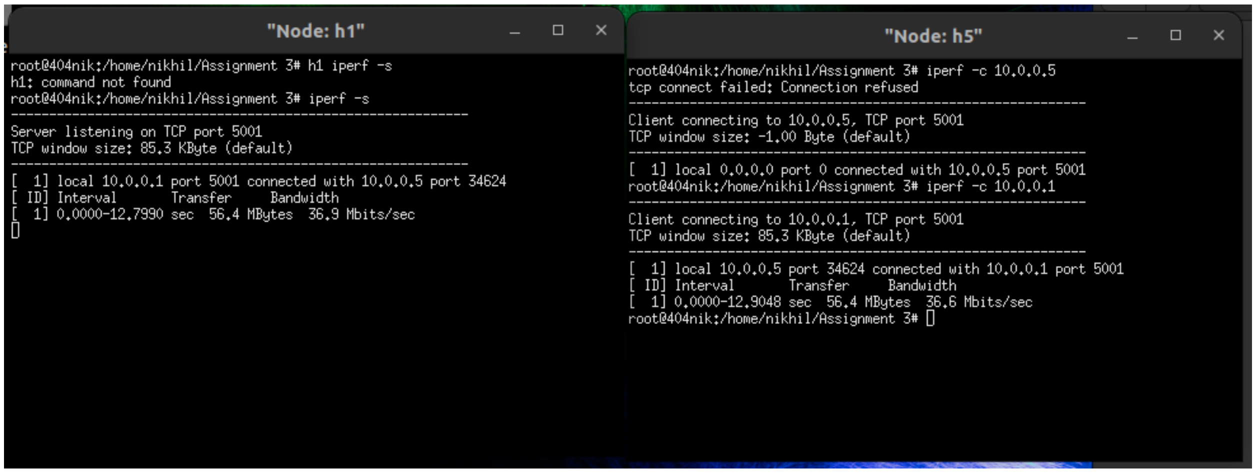
```
mininet> h4 ping -c 3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=103 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=100 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=100 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 100.355/101.213/102.875/1.175 ms
mininet> h4 ping -c 3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=100 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=100 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 100.277/100.688/101.472/0.554 ms
mininet> h4 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=101 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=100 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=100 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 100.322/100.610/101.154/0.384 ms
mininet> h4 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=287 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=144 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=144 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 144.032/191.698/286.618/67.118 ms
```

Ping each other



```
mininet> h5 ping -c 3 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=107 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=106 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=105 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 104.803/105.961/106.817/0.849 ms
mininet> h5 ping -c 3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=103 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=103 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=103 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 102.723/102.914/103.120/0.162 ms
mininet> h5 ping -c 3 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=105 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=103 ms

64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=103 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 102.584/103.402/104.844/1.022 ms
mininet> h5 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=144 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=144 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=142 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 141.983/143.265/143.932/0.906 ms
```

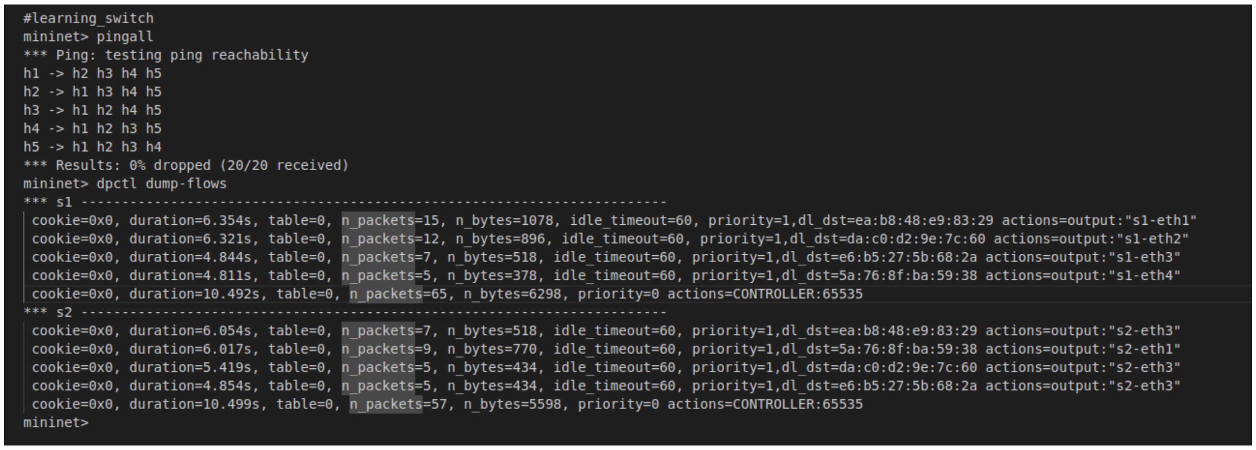
Ping each other



```
root@404nik:/home/nikhil/Assignment 3# h1 iperf -s
h1: command not found
root@404nik:/home/nikhil/Assignment 3# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.1 port 5001 connected with 10.0.0.5 port 34624
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-12.7990 sec  56.4 MBytes  36.9 Mbits/sec
```

```
root@404nik:/home/nikhil/Assignment 3# iperf -c 10.0.0.5
tcp connect failed: Connection refused
------------------------------------------------------------
Client connecting to 10.0.0.5, TCP port 5001
TCP window size: -1.00 Byte (default)
------------------------------------------------------------
[  1] local 0.0.0.0 port 0 connected with 10.0.0.5 port 5001
root@404nik:/home/nikhil/Assignment 3# iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.5 port 34624 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-12.9048 sec  56.4 MBytes  36.6 Mbits/sec
root@404nik:/home/nikhil/Assignment 3# 
```

Bandwidth h1_h5 in learning hub



```
#learning_switch
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=6.354s, table=0, n_packets=15, n_bytes=1078, idle_timeout=60, priority=1,dl_dst=ea:b8:48:e9:83:29 actions=output:"s1-eth1"
 cookie=0x0, duration=6.321s, table=0, n_packets=12, n_bytes=896, idle_timeout=60, priority=1,dl_dst=da:c0:d2:9e:7c:60 actions=output:"s1-eth2"
 cookie=0x0, duration=4.844s, table=0, n_packets=7, n_bytes=518, idle_timeout=60, priority=1,dl_dst=e6:b5:27:5b:68:2a actions=output:"s1-eth3"
 cookie=0x0, duration=4.811s, table=0, n_packets=5, n_bytes=378, idle_timeout=60, priority=1,dl_dst=5a:76:8f:ba:59:38 actions=output:"s1-eth4"
 cookie=0x0, duration=10.492s, table=0, n_packets=65, n_bytes=6298, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=6.054s, table=0, n_packets=7, n_bytes=518, idle_timeout=60, priority=1,dl_dst=ea:b8:48:e9:83:29 actions=output:"s2-eth3"
 cookie=0x0, duration=6.017s, table=0, n_packets=9, n_bytes=770, idle_timeout=60, priority=1,dl_dst=5a:76:8f:ba:59:38 actions=output:"s2-eth1"
 cookie=0x0, duration=5.419s, table=0, n_packets=5, n_bytes=434, idle_timeout=60, priority=1,dl_dst=da:c0:d2:9e:7c:60 actions=output:"s2-eth3"
 cookie=0x0, duration=4.854s, table=0, n_packets=5, n_bytes=434, idle_timeout=60, priority=1,dl_dst=e6:b5:27:5b:68:2a actions=output:"s2-eth3"
 cookie=0x0, duration=10.499s, table=0, n_packets=57, n_bytes=5598, priority=0 actions=CONTROLLER:65535
mininet>
```

Pingall and installed rules (learning switch)

➔Firewall Monitor (firewall_monitor.py) -

- ◆ The code provided implements a firewall and monitoring functionality in an SDN (Software-Defined Networking) controller using the Ryu framework. It restricts communication between specific hosts and counts the packets coming from a specific host.
- ◆ Here's an explanation of the code:

  - Import necessary modules and libraries, including the sys module, collections, Ryu-related modules, and packet-related modules.

- Check the Python version to determine whether to import MutableMapping from collections.abc or just collections. This is done to handle differences in Python versions.

- Define a class named Controller that extends RyuApp. This class represents the SDN controller application.

- Specify the OpenFlow protocol version(s) supported by the controller, which is set to OpenFlow 1.3.

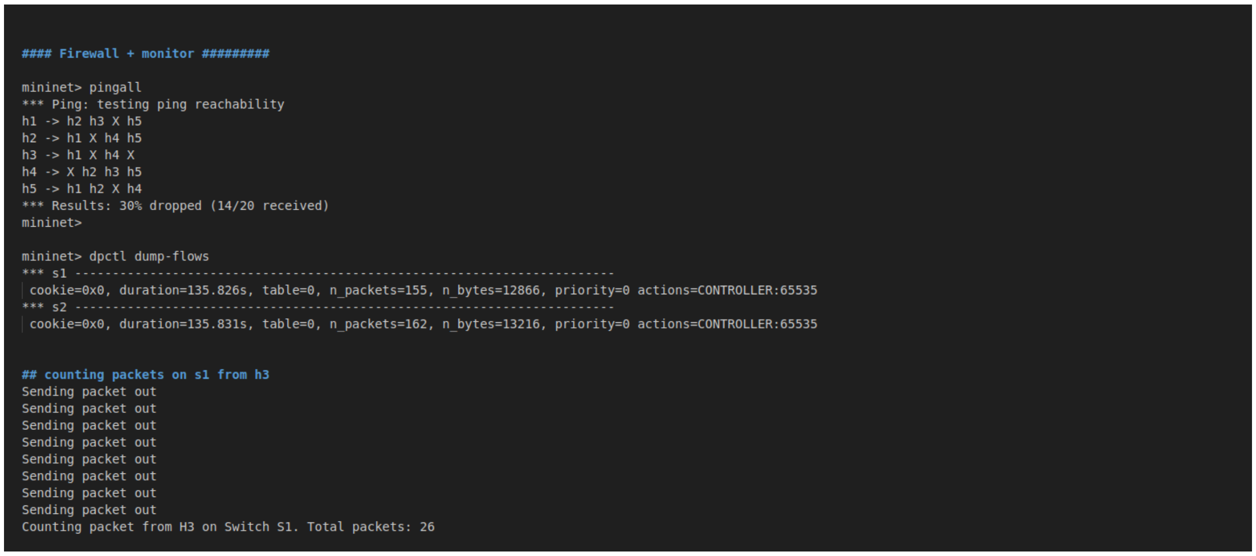
- In the __init__ method, initialize the controller application. It creates a dictionary called packet_count to store packet counts.

- The features_handler method is called when a switch connects to the controller (during the handshake). It sets up an initial flow entry to send packets to the controller and initializes a flow table entry for packet counting.

- The packet_in_handler method processes incoming packets from the switch:

- It extracts the source and destination MAC addresses from the packet.
- It defines MAC addresses for hosts H2, H3, H5, H1, and H4.
- For packets received on Switch S1 with the source MAC address "00:00:00:00:00:03" (H3), it counts the packets and logs the count.
- It blocks communication between H2 and H3 with H5 and H1 with H4 by checking the source and destination MAC addresses. If a packet matches these conditions, it logs that communication is blocked.
- For all other packets, it allows communication and sends the packet out to all ports (except the input port).
- The __add_flow method is used to add flow table entries to the switch, allowing the controller to define how packets should be forwarded based on certain criteria.

- So code implements a firewall and packet monitoring mechanism for an SDN network. It restricts communication between specific hosts while counting packets coming from H3 on Switch S1. Any communication that is not blocked is allowed to continue, and the packet counts are logged



```
#### Firewall + monitor #########

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X h5
h2 -> h1 X h4 h5
h3 -> h1 X h4 X
h4 -> X h2 h3 h5
h5 -> h1 h2 X h4
*** Results: 30% dropped (14/20 received)
mininet>

mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=135.826s, table=0, n_packets=155, n_bytes=12866, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=135.831s, table=0, n_packets=162, n_bytes=13216, priority=0 actions=CONTROLLER:65535


## counting packets on s1 from h3
Sending packet out
Sending packet out
Sending packet out
Sending packet out
Sending packet out
Sending packet out
Sending packet out
Sending packet out
Counting packet from H3 on Switch S1. Total packets: 26
```

➔Load Balancer (load_balancer.py) -

imports and Ryu App Initialization:

The code starts with importing necessary modules from Ryu, such as the RyuApp class, event handling components, OpenFlow protocol versions, and packet manipulation libraries.
It defines a class named LoadBalancingSwitch that inherits from RyuApp. This class represents the load balancer application.
Server IPs and MACs:

The load balancer defines the IP and MAC addresses for the two servers (h4 and h5) it is balancing traffic between.
It also defines a virtual IP address (virtual_ip) that the clients will connect to.
IP Mapping:

It maintains a dictionary ip_to_mac that maps IP addresses to MAC addresses of the hosts in the network.
It also maintains a ip_to_port dictionary that maps server IP addresses to the corresponding OpenFlow port on the switch.
Initialization:

In the __init__ method, the load balancer initializes the current_ip and next_ip variables. Initially, the current_ip is set to h4's IP, and the next_ip is set to h4's IP. The load balancer starts by sending traffic to h4.

Packet In Handler:

The packet_in_handler method is decorated with
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER), indicating that it handles incoming OpenFlow packet-in events.
When an ARP request packet is received, this method processes it.
ARP Request Handling:

When an ARP request packet is received from a client (h1, h2, h3), the load balancer adds flow entries to the switch to establish the correct mapping of IP addresses and MAC addresses.
It calls add_client_server_flow to add flow entries that direct traffic from the client to the appropriate server.
The load balancer also sends an ARP response to the client with the virtual IP address, ensuring that the client maps the virtual IP to the real MAC address of the chosen server (h4 or h5).
The current_ip is updated to switch between h4 and h5, ensuring even load distribution.
Flow Entry Setup:

The add_client_server_flow method sets up flow entries for packets from clients to servers. It modifies the destination IP address and forwards the packets to the appropriate server.
ARP Response Handling:

The send_response method constructs and sends an ARP response to the client with the virtual IP address and the selected server's MAC address.
Load Balancing Logic:

The load balancing is achieved by alternating the choice of the server (h4 and h5) for each new client ARP request. The current_ip is switched between the two servers.
OpenFlow Message Handling:

The load balancer uses OpenFlow to communicate with the switch. It constructs OpenFlow messages to add flow entries and send packets out to the switch.

➔ Answers to Part A

Latency Values:

For Controller Type "controler_hub":

h1 to h2: 62.3 ms, 62.4 ms, 65.0 ms (avg: 63.566 ms)

h1 to h3: 62.365 ms, 62.4 ms, 66.2 ms (avg: 63.988 ms)
h1 to h4: 106 ms, 107 ms, 108 ms (avg: 107.0 ms)
h1 to h5: 106 ms, 108 ms, 107 ms (avg: 107.0 ms)
For Controller Type "learning_switch":

h1 to h2: 60.2 ms, 60.5 ms, 132 ms (avg: 84.566 ms)
h1 to h3: 61.8 ms, 62.2 ms, 127 ms (avg: 83.0 ms)
h1 to h4: 102 ms, 104 ms, 216 ms (avg: 140.667 ms)
h1 to h5: 106 ms, 107 ms, 210 ms (avg: 141.0 ms)
Observations:

For the "controler_hub" setup, the average latency is higher, especially between h1 and h4 and h1 and h5. This is because all traffic is being sent to the controller, which adds to the latency.
For the "learning_switch" setup, latency is generally lower due to the switches making forwarding decisions independently without involving the controller.
Throughput:

Controller: Hub Controller

Bandwidth between h1 and h5: ~29.5 Mbits/sec
Bandwidth between h5 and h1: ~29 Mbits/sec
All pings between hosts have varying latencies, generally in the range of 60-216 ms.
Flow rules on switches show that all traffic is being forwarded to the controller.
Controller: Learning Switch

Bandwidth between h1 and h5: ~35.6 Mbits/sec
Bandwidth between h5 and h1: ~35.2 Mbits/sec
All pings between hosts have lower and more consistent latencies in the range of 60-132 ms.
Flow rules on switches show that the Learning Switch is effectively forwarding traffic based on MAC addresses, which leads to better performance.
The reason for the difference in throughput and latency between the Hub Controller and Learning Switch lies in their operation:

Hub Controller: In this scenario, all traffic is being sent to the controller (actions=CONTROLLER:65535). This means the controller has to process every packet and make forwarding decisions. This introduces latency due to the additional controller processing time, leading to higher latencies and slightly lower throughput.

Learning Switch: In this scenario, the Learning Switch has learned the MAC addresses of hosts by observing traffic, and it can make forwarding decisions without sending packets to the controller. This results in faster and more efficient packet forwarding, leading to lower latencies and higher throughput.

Overall, the Learning Switch operates more efficiently and performs better because it can make local forwarding decisions based on the learned MAC addresses, reducing the need to involve the controller for every packet.

For the "controler_hub" setup, the switches have rules that forward all packets to the controller, as indicated by the "CONTROLLER:65535" actions in the flow entries.

For the "learning_switch" setup, the switches have installed flow entries based on the learned MAC addresses. They have entries for specific destination MAC addresses to forward packets to the respective output port. The entries include the Ethernet destination addresses and the output port for each.

Both controller types seem to be functioning as expected. The "controler_hub" setup forwards all traffic to the controller, while the "learning_switch" setup allows switches to learn and make independent forwarding decisions based on the learned MAC addresses.

➔ Answers to Part B

Ping Results:

h1 can ping h2 and h5 successfully.
h2 can ping h1 but not h4 and h5.
h3 can ping h1 but not h4.
h4 can ping h2 and h3 but not h5.
h5 can ping h1 and h2.
Results Analysis:
The ping results indicate that some hosts can communicate with each other while others cannot. This is likely due to the firewall rules set in place. For example, hosts h1, h2, and h5 can communicate with each other because they have established rules. On the other hand, h3 and h4 are not able to communicate with certain hosts due to missing or blocking rules.

Installed Rules:

On switch s1, there is a rule with actions=CONTROLLER:65535, indicating that all traffic is being sent to the controller. There are 155 packets processed on s1.
On switch s2, there is a similar rule with actions=CONTROLLER:65535, indicating that all traffic is also being sent to the controller. There are 162 packets processed on s2.
Ways to Minimize Firewall Rules:
To minimize the number of firewall rules on the switch and improve network performance, you can:

Implement specific allow rules: Instead of sending all traffic to the controller, you can configure specific rules to allow the desired traffic between hosts. For example, you can set rules to allow traffic between all hosts or based on specific port numbers.

Use wildcard rules: You can create wildcard rules that cover multiple hosts or subnets, reducing the number of individual rules needed.

Apply default deny policy: Instead of explicitly allowing traffic, you can configure a default deny policy, which drops all traffic by default and then explicitly allows only the desired traffic.

Real-time Firewall Policies:
If the network operator intends to implement real-time firewall policies, they can utilize SDN controllers and OpenFlow to dynamically update firewall rules. This allows them to adapt to changing network conditions and security requirements in real-time.

For example, the controller can monitor the network and adjust firewall rules based on observed traffic patterns or security events. When specific conditions are met, the controller can add, modify, or remove rules to enforce the desired policies in real-time. This dynamic approach ensures that the network remains secure and responsive to changing requirements.

➔ Answers to part C

we can follow these additional steps without changing the provided code:

Server Load Monitoring: You need a mechanism to monitor the current load on each server. This can be done using various metrics such as CPU usage, memory usage, or the number of active connections. You can use tools like SNMP (Simple Network Management Protocol) to gather server load data.

Load Balancer Decision Logic: Implement a decision logic within your load balancer that takes into account the current server loads. This logic can use the load metrics collected in step 1 to determine which server is the least loaded or the most suitable to handle new requests. The decision logic could be as simple as selecting the server with the lowest load, or it can be more complex, considering factors like server capacity or response time.

Dynamic Server Selection: Modify the load balancer code to dynamically select the appropriate server based on the decision logic. When a new client request arrives, the load balancer should choose the server that best meets the load balancing policy criteria.

Periodic Load Checks: Implement a mechanism to periodically check the server loads and update the load balancing decision accordingly. You can set a timer or schedule regular load checks to ensure the load balancing decision remains up to date.

Health Checks: Along with load monitoring, implement health checks to ensure that the selected server is still available and responsive. If a server becomes unhealthy, the load balancer should automatically reassign requests to other healthy servers.

Feedback Mechanism: Optionally, implement a feedback mechanism to gather statistics about server performance and load distribution. This feedback can be used to refine and optimize the load balancing algorithm over time.

Configurability: Provide options for configuring the load balancing policy and thresholds. Different applications may require different load balancing strategies, so make sure your load balancer is configurable to accommodate various use cases.

Remember that load balancing policies can be tailored to specific application requirements. Depending on your use case, you may need to prioritize low latency, evenly distributed load, or some other criteria. The key is to continuously monitor and adapt the load balancing strategy to meet your application's needs efficiently and effectively.

➔Working of code (commands) -

Run Ryu application - ryu-manager ./<py_file>.py
Run mininet topology - sudo mn --custom topology.py --topo customTopology --controller remote